IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

GUY DUPENLOUP

Serial No.: 09/026,790

Filed: February 20, 1998

For: AUTOMATIC SYNTHESIS SCRIPT
      GENERATION FOR SYNOPSIS DESIGN
      COMPILER

Group Art Unit: 2768

Examiner: A. Thompson

APPELLANT'S BRIEF
ON APPEAL TO THE BOARD OF PATENT APPEALS AND INTERFERENCES

Commissioner of Patents and Trademarks
Washington, D.C. 20231

Dear Sir:

Appellant in the above-captioned patent application appeals the final rejection of claims 1 to 6 and 9 to 20 set forth in the Office Action dated March 21, 2000, a timely Notice of Appeal having been filed on June 21, 2000.

0262018.1

I. REAL PARTY IN INTEREST

The real party in interest in this application is LSI Logic Corporation, a Delaware corporation, having a place of business at 1551 McCarthy Boulevard, Milpitas, California 95035, pursuant to an assignment which was recorded at reel 9366, frame 0808 on July 6, 1998.

II. RELATED APPEALS AND INTERFERENCES

Appellant is unaware of any related appeals or interferences.

III. STATUS OF CLAIMS

Claims 1 to 6 and 9 to 20 have been finally rejected and are the subject matter of this Appeal. Claims 7 and 8 have been canceled. In accordance with 37 C.F.R. § 1.192(c)(9), a copy of the claims involved in this appeal is included in the Appendix attached hereto.

IV. STATUS OF THE AMENDMENTS

No Amendments have been filed subsequent to the final rejection.

V. SUMMARY OF THE INVENTION

The present invention is directed to generation of synthesis scripts to synthesize integrated circuit designs into a gate-level description by identifying from a generic netlist: hardware elements, key pins for identified hardware elements, and/or design structure and hierarchy.

Due to the complexity of modern integrated circuits, integrated circuit design has become a fairly structured multi-step process. Typically, one of the early steps in this process is for a group of design engineers to specify the circuit design in a hardware description language (HDL). Generally, HDL can be thought of as the hardware design

-2-

equivalent of a software programming language. Due to the inherent differences between hardware and software, however, HDL is typically made flexible enough to be capable of specifying a design at various levels of abstraction, such as gate level, register transfer level (RTL), boolean level or algorithmic level.

Once a design has been specified using HDL, the next major step, typically, is to compile the HDL code using a logic synthesis tool and an input technology library in order to obtain a netlist for the integrated circuit. More specifically, the output of the synthesis tool typically is a list of actual specific circuit components and a list of the interconnections between those components, collectively called a "netlist". Unlike HDL, which is mostly human-generated and therefore very human readable, a netlist is usually just a long list from which it is often difficult for humans to obtain any meaningful information.

Initially, logic synthesis tools were configured so as to perform the same synthesis algorithms irrespective of the circuit design. However, it was eventually realized that more optimal synthesis results often could be obtained by tailoring the synthesis operations to the specific design in question. As a result, conventional techniques began to adaptively generate scripts for driving the synthesis tool based on the input HDL code. See, for example, U.S. Patent 5,812,416 (Gupte).

While such techniques often are much better than simply using generic synthesis scripts, Appellant has discovered a significant number of design issues that can be missed when directly analyzing HDL code. See, for example, page 41, line 18 to page 45, line 4 of the Specification. Appellant further recognized that synthesis tools typically generate a generic netlist as an intermediate step to producing a final technology-dependent netlist from the input HDL code and that analysis of the generic netlist will identify many design issues that are missed when only analyzing HDL code. Techniques for generic netlist analysis are discussed throughout the Specification, particularly from page 81, line 20 to page 98, line 21.

-3-

Generally speaking, the components specified in a generic netlist are not associated with any specific technology, but rather represent standard functionality, such as a generic AND gate, a generic OR gate, or a generic Multiplexer circuit. Once a technology has been decided upon, such generic components are converted into actual components based on drive-strength requirements, timing considerations, and similar design issues.

By identifying hardware elements and/or key pins for identified hardware elements from a generic netlist, the present invention often can provide more optimal synthesis scripts than are possible with the conventional techniques. As a result, more effective synthesis can be performed, thereby significantly reducing subsequent design correction efforts.

VI. ISSUES PRESENTED ON APPEAL

The issue is whether claims 1 to 6 and 9 to 20 are properly rejected under 35 U.S.C. § 102(e) over U.S. Patent 5,812,416 (Gupte).

VII. GROUPING OF THE CLAIMS

In the Office Action, the Examiner grouped all of the pending claims into a single group. However, Appellant believes that the claims are more appropriately grouped as follows:

GROUP 1: Claims 1 to 6, 11 to 13, and 15 to 17

GROUP 2: Claims 9, 10, 14, and 18 to 20

VIII. ARGUMENT

DISCUSSION OF ISSUES ON APPEAL

Anticipation under § 102 is a strict word-for-word identity test in which a single prior art reference must show, with word-for-word identity every element of the invention. The test has been stated as follows:

> "For a prior art reference to anticipate in terms of 35 U.S.C. § 102, every element in the claimed invention must be shown in a single reference."

In re Bond, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990) (quoting Diversitech, 7 U.S.P.Q.2d 1315). Moreover, the Federal Circuit has held that:

> "The identical invention must be shown in as complete detail as is contained in the . . . claim."

Richardson v. Suzuki Motor Co., 868 F.2d 1226, 1236, 9 U.S.P.Q.2d 1913, 1920, (Fed. Cir. 1989).

As discussed below, the above word-for-word identity tests for establishing anticipation under § 102 has not been met for any of the following groups of claims.

Group 1 Claims

Claims 1 to 6 and 15 are directed to generating synthesis scripts to synthesize integrated circuit (IC) designs from a generic netlist description into a gate-level description. Initially, hardware elements are identified in the generic netlist and key pins are determined for each of the identified hardware elements. Design structure and hierarchy also are extracted from the generic netlist. Then, script is generated to cause a logic synthesis tool to apply bottom-up synthesis and top-down characterization to

modules and sub-modules of the IC design and to cause a logic synthesis tool to repeat such bottom-up and top-down applications until constraints are satisfied.

Claims 11 to 13, 16 and 17 are directed to generating synthesis scripts to synthesize integrated circuit (IC) designs in a RTL level description into a gate-level description. Initially, hardware elements are identified in the generic netlist and key pins are determined for each of the identified hardware elements. Critical design structure and hierarchy also are extracted from the generic netlist. Then, bottom-up synthesis and top-down characterization are applied to modules and sub-modules of the IC design, and such bottom-up and top-down applications are repeated until constraints are satisfied. Finally, design compile scripts are created to synthesize modules and sub-modules and the IC design having such satisfied constraints.

The foregoing combinations of features are not disclosed by the applied art. In particular, Gupte does not disclose at least the features of: (i) identifying hardware elements in a generic netlist; (ii) determining key pins for each of such identified hardware elements; or (iii) extracting design structure and hierarchy from a generic netlist.

With respect to feature (i), the Examiner has cited column 14, lines 12 to 17 of Gupte. It is noted that this portion of Gupte merely discusses parsing <u>HDL code</u> and says nothing at all about a generic netlist.

In stating the rejection, the Examiner includes the words "generic netlist" in parentheses after a reference to Gupte's processing of HDL code. Thus, the Examiner apparently is equating HDL code with a generic netlist. However, the Examiner has cited nothing in Gupte that indicates any such equivalence and has provided no other basis for asserting that HDL code is the same as a generic netlist. Moreover, Appellant has reviewed Gupte in detail and is unable to find anything that indicates that the two are the same.

In response to Appellant's assertions that HDL code is significantly different than a generic netlist, the Examiner has merely asserted that HDL code is "generic", without

addressing the more relevant issue as to whether HDL code is the same as a "generic netlist". Clearly, the two are not the same. In fact, those skilled in the art recognize that a netlist typically is generated only after synthesizing HDL code. This process is even described in Gupte -- from column 13, line 62 to column 14, line 3. Specifically, that portion of Gupte notes that synthesis scripts, HDL code, constraints files and technology libraries are input into the synthesis tool and, from these inputs, the synthesis tool produces synthesized gate-level netlists.

Thus, rather than indicating that HDL code is the same as a generic netlist, Gupte actually teaches that a netlist is generated from HDL code by using a synthesis process. Accordingly, Gupte clearly does not disclose the feature of identifying hardware elements in a generic netlist and, therefore, could not possibly have anticipated the Group 1 claims.

The Examiner apparently also relies on column 14, lines 12 to 17, of Gupte as showing the feature of determining key pins in each of such identified hardware elements (feature (ii) above). In particular, the Examiner asserts that "Parsing inherently involves determining key pins for identified hardware elements." However, it is noted that nothing in Gupte indicates that his parsing of HDL inherently involves determining key pins for identified hardware elements. In addition, the Examiner has cited no other art that would indicate that parsing HDL code inherently includes this feature of the invention. In this regard, the law is clear that:

> "To establish inherency, the extrinsic evidence [emphasis
> added] must make clear that the missing descriptive matter
> is necessarily present in the thing described in the reference,
> and that it would be so recognized by persons of ordinary
> skill. [citations omitted]. Inherency, however, may not be
> established by probabilities or possibilities. The mere fact
> that a certain thing may result from a given set of
> circumstances is not sufficient. [citations omitted]"

In re Robertson, (Fed. Cir. 1999) 169 F.3d 743, 745, 49 U.S.P.Q.2d 1949.

Because the Examiner's assertion of inherence is not supported by extrinsic evidence, it cannot be said that feature (ii) of the Group 1 claims is disclosed by Gupte.

Moreover, the Examiner's assertions with respect to feature (ii) also rely on the equivalence of HDL code and a generic netlist. However, as noted above, such equivalence has not been established. For this additional reason, feature (ii) of the Group 1 claims is not disclosed by Gupte.

The Examiner relies on column 14, lines 12 to 22 of Gupte as showing the feature of extracting design structure and hierarchy from a generic netlist (feature (iii) above). However, while this portion of Gupte discusses creation of certain data structures, nothing therein refers to extracting design structure or hierarchy from a generic netlist. Thus, feature (iii) of the Group 1 claims also appears not to be disclosed by Gupte.

In addition, the Examiner is apparently once again relying on the equivalence between HDL code and a generic netlist, which equivalence has not been shown. For this additional reason, the feature of extracting design structure and hierarchy from a generic netlist also is not disclosed by Gupte.

As indicated above, Gupte fails to disclose several important features of the present invention. In view of the standards cited above for an anticipation rejection under § 102, it is clear that Gupte could not possibly have anticipated the Group 1 claims.

Group 2 Claims

Claims 9, 10, 14, and 18 to 20 are directed to generating synthesis scripts to synthesize integrated circuit (IC) designs in a RTL level description into a gate-level description. Initially, key pins for identified hardware elements are determined from a generic netlist. Also, critical design structure and hierarchy are extracted from the generic netlist. Bottom-up synthesis and top-down characterization are applied to modules and sub-modules of the IC design, and then such bottom-up and top-down

-8-

applications are repeated until constraints are satisfied. Finally, design compile scripts are created to synthesize modules and sub-modules and the IC design having such satisfied constraints.

The foregoing combination of features is not disclosed by the applied art. In particular, Gupte does not disclose at least the features of: (i) determining key pins for identified hardware elements from a generic netlist; or (ii) extracting critical design structure and hierarchy from the generic netlist.

With respect to feature (i), the Examiner simply asserts that "Parsing inherently involves determining key pins for identified hardware elements." It is assumed that the Examiner is referring to column 14, lines 12 to 17, of Gupte. However, for the reasons set forth in detail above, the requirements for showing such inherency set forth by the Federal Circuit in In re Robertson clearly have not been satisfied in this case.

In addition, this portion of Gupte says nothing at all about determining anything from a generic netlist, but rather only discusses parsing HDL code. As a result, Gupte clearly would not have disclosed the feature of determining key pins for identified hardware elements from a generic netlist.

The Examiner cites column 14, lines 12 to 22, of Gupte as showing the feature of extracting critical design structure and hierarchy from a generic netlist (feature (ii) above). However, while this portion of Gupte discusses creation of certain data structures, nothing therein refers to extracting design structure or hierarchy from a generic netlist. Thus, feature (ii) of the Group 2 claims also appears not to be disclosed by Gupte.

In addition, in making this rejection the Examiner is apparently once again relying on the equivalence between HDL code and a generic netlist, which equivalence has not been shown. For this additional reason, the feature of extracting critical design structure and hierarchy from the generic netlist also is not disclosed by Gupte.

As indicated above, Gupte fails to disclose several important features of the Group 2 claims of the present invention. In view of the standards cited above for an

0262018.1

anticipation rejection under § 102, it is clear that Gupte could not possibly have anticipated the Group 2 claims.

CONCLUDING REMARKS

As Appellant has shown above, for a number of different reasons, nothing in the Gupte discloses the invention recited by the claims on appeal. Appellant therefore respectfully submits that the claimed invention is patentably distinct over the applied art.
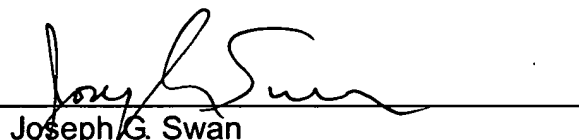
In view of the foregoing remarks, Appellant respectfully requests that the rejection of claims 1 to 6 and 9 to 20 be reversed and a Notice of Allowance issued.

Respectfully submitted,

MITCHELL, SILBERBERG & KNUPP LLP

Dated: August 21, 2000   By_____

Joseph G. Swan
Registration No. 41,338

MITCHELL, SILBERBERG & KNUPP LLP
11377 West Olympic Boulevard
Los Angeles, California 90064
Telephone: (310) 312-2000
Facsimile: (310) 312-3100

0262018.1

APPENDIX

Claims on Appeal

1.   A method of generating synthesis scripts to synthesize integrated circuit (IC) designs from a generic netlist description into gate-level description, said method comprising the steps of:

identifying hardware elements in the generic netlist;

determining key pins for each of said identified hardware elements;

extracting design structure and hierarchy from the Generic netlist;

generating script to cause a logic synthesis tool to apply bottom-up synthesis to modules and sub-modules of the IC design;

generating script to cause a logic synthesis tool to apply top-down characterization to modules and sub-modules of the IC design; and

generating script to cause a logic synthesis tool to repeat said bottom-up and said top-down applications until constraints are satisfied.

2.   A method according to claim 1 wherein said step of extracting design structure allows for a multilevel structuring of modules of the IC design.

3.   A method according to claim 1 further comprising the step of generating script to cause a logic synthesis tool to apply initial mapping to the IC design.

4.   A method according to claim 1 wherein the logic synthesis tool is Synopsys Design Compiler.

5.   A method according to claim 1 further comprising the step of rearranging design hierarchy by changing the design.

0262018.1

6. A method according to claim 1 further comprising the step of generating script to cause a logic synthesis tool to ungroup modules of the IC design.

7. [Canceled]

8. [Canceled]

9. An apparatus for generating synthesis scripts to synthesize integrated circuit (IC) designs in RTL level description into gate-level description, comprising:

a processor;

memory connected to said processor;

said memory having instructions for said processor to

determine key pins for identified hardware elements from a generic netlist;

extract critical design structure and hierarchy from the generic netlist;

apply bottom-up synthesis to modules and sub-modules of the IC design;

apply top-down characterization to modules and sub-modules of the IC design;

repeat said bottom-up and said top-down applications until constraints are satisfied; and

create design compile scripts to synthesize modules and sub-modules and the IC design having said satisfied constraints.

10. An apparatus for generating synthesis scripts to synthesize integrated circuit (IC) designs in RTL level description into gate-level description, comprising:

means for determining key pins for identified hardware elements from a generic netlist;

means for extracting critical design structure and hierarchy from the generic netlist;

means for applying bottom-up synthesis to modules and sub-modules of the IC design;

means for applying top-down characterization to modules and sub-modules of the IC design;

means for repeating said bottom-up and said top-down applications until constraints are satisfied; and

means for creating design compile scripts to synthesize modules and sub-modules and the IC design having said satisfied constraints.

11.   A computer storage medium containing instructions for generating synthesis scripts to synthesize integrated circuit (IC) designs in RTL level description into gate-level description, said instructions comprising the steps of:

identifying hardware elements in the generic netlist;

determining key pins for each of said identified hardware elements;

extracting critical design structure and hierarchy from the generic netlist;

applying bottom-up synthesis to modules and sub-modules of the IC design;

applying top-down characterization to modules and sub-modules of the IC design;

repeating said bottom-up and said top-down applications until constraints are satisfied; and

creating design compile scripts to synthesize modules and sub-modules and the IC design having said satisfied constraints.

12.   A computer storage medium of claim 11 wherein said computer storage medium is selected from a group consisting of magnetic device, optical device, magneto-optical device, floppy diskette, CD-ROM, magnetic tape, computer hard drive, and memory card.

13.   A process for generating synthesis scripts to synthesize integrated circuit (IC) designs in RTL level description into gate-level description, said process comprising the steps of:

identifying hardware elements in the generic netlist;

determining key pins for each of said identified hardware elements;

extracting critical design structure and hierarchy from the generic netlist;

applying bottom-up synthesis to modules and sub-modules of the IC design;

applying top-down characterization to modules and sub-modules of the IC design;

repeating said bottom-up and said top-down applications until constraints are satisfied; and

creating design compile scripts to synthesize modules and sub-modules and the IC design having said satisfied constraints.

14.   A computer system for generating synthesis scripts to synthesize integrated circuit (IC) designs in RTL level description into gate-level description, said system comprising:

means for determining key pins for identified hardware elements from a generic netlist;

means for extracting critical design structure and hierarchy from the generic netlist;

means for applying bottom-up synthesis to modules and sub-modules of the IC design;

means for applying top-down characterization to modules and sub-modules of the IC design;

means for repeating said bottom-up and said top-down applications until constraints are satisfied; and

means for creating design compile scripts to synthesize modules and sub-modules and the IC design having said satisfied constraints.

15.   A method according to Claim 1, wherein I/O conditions and constraints of the modules of the IC design captured during the top-down characterization are used to re-optimize the IC design during the bottom-up synthesis.

16.   A computer storage medium according to Claim 11, wherein I/O conditions and constraints of the modules of the IC design captured during the top-down characterization are used to re-optimize the IC design during the bottom-up synthesis.

17.   A process according to Claim 13, wherein I/O conditions and constraints of the modules of the IC design captured during the top-down characterization are used to re-optimize the IC design during the bottom-up synthesis.

18.   An apparatus according to Claim 9, wherein I/O conditions and constraints of the modules of the IC design captured during the top-down characterization are used to re-optimize the IC design during the bottom-up synthesis.

19.   An apparatus according to Claim 10, wherein I/O conditions and constraints of the modules of the IC design captured during the top-down characterization are used to re-optimize the IC design during the bottom-up synthesis.

20.   A computer system according to Claim 14, wherein I/O conditions and constraints of the modules of the IC design captured during the top-down characterization are used to re-optimize the IC design during the bottom-up synthesis.

0262018.1